

# Polymorphism, Subtyping, and Type Inference in MLsub

---

Stephen Dolan and Alan Mycroft

January 18, 2017

Computer Laboratory  
University of Cambridge

# The select function

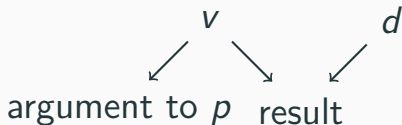
`select p v d = if (p v) then v else d`

In ML, select has type scheme

$$\forall \alpha. (\alpha \rightarrow \text{bool}) \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha$$

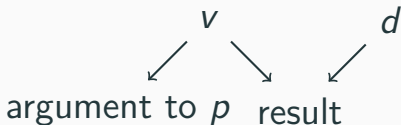
# Data flow in select

`select p v d = if (p v) then v else d`



# Data flow in select

`select p v d = if (p v) then v else d`



In MLsub, `select` has this type scheme:

$$\forall \alpha, \beta. (\alpha \rightarrow \text{bool}) \rightarrow \alpha \rightarrow \beta \rightarrow (\alpha \sqcup \beta)$$

# The MLsub Type System

---

$$\Gamma \vdash e : \tau$$

$\Gamma \vdash e : \tau$

# Expressions of MLsub

We have functions

$x$        $\lambda x.e$        $e_1 e_2$

... and records

$\{\ell_1 = e_1, \dots, \ell_n = e_n\}$        $e.l$

... and booleans

$\text{true}$        $\text{false}$        $\text{if } e_1 \text{ then } e_2 \text{ else } e_3$

... and let

$\hat{x}$        $\text{let } \hat{x} = e_1 \text{ in } e_2$



$\Gamma \vdash e : \tau$

# Typing rules of MLsub

MLsub is

ML +

$$\text{(SUB)} \quad \frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash e : \tau_2} \quad \tau_1 \leq \tau_2$$

$\Gamma \vdash e : \tau$

# Constructing Types

The standard definition of types looks like:

$$\tau ::= \perp \mid \tau \rightarrow \tau \mid \top$$

(ignoring records and booleans for now)

# Constructing Types

The standard definition of types looks like:

$$\tau ::= \perp \mid \tau \rightarrow \tau \mid \top$$

(ignoring records and booleans for now)

with a subtyping relation like:

$$\frac{}{\perp \leq \tau} \quad \frac{}{\tau \leq \top} \quad \frac{\tau'_1 \leq \tau_1 \quad \tau_2 \leq \tau'_2}{\tau_1 \rightarrow \tau_2 \leq \tau'_1 \rightarrow \tau'_2}$$

# Lattices

These types form a lattice:

- least upper bounds  $\tau_1 \sqcup \tau_2$
- greatest lower bounds  $\tau_1 \sqcap \tau_2$

# Lattices

These types form a lattice:

- least upper bounds  $\tau_1 \sqcup \tau_2$
- greatest lower bounds  $\tau_1 \sqcap \tau_2$

$$\frac{e_1 : \tau_1 \quad e_2 : \tau_2}{\text{if rand } () \text{ then } e_1 \text{ else } e_2 : \tau_1 \sqcup \tau_2}$$

# Bizarrely difficult questions

Is this true, for all  $\alpha$ ?

$$\alpha \rightarrow \alpha \leq \perp \rightarrow \top$$



# Bizarrely difficult questions

Is this true, for all  $\alpha$ ?

$$\alpha \rightarrow \alpha \leq \perp \rightarrow \top$$

How about this?

$$(\perp \rightarrow \top) \rightarrow \perp \leq (\alpha \rightarrow \perp) \sqcup \alpha$$

# Bizarrely difficult questions

Is this true, for all  $\alpha$ ?

$$\alpha \rightarrow \alpha \leq \perp \rightarrow \top$$

How about this?

$$(\perp \rightarrow \top) \rightarrow \perp \leq (\alpha \rightarrow \perp) \sqcup \alpha$$

Yes, it turns out, by **case analysis** on  $\alpha$ .

# Bizarrely difficult questions

Is this true, for all  $\alpha$ ?

$$\alpha \rightarrow \alpha \leq \perp \rightarrow \top$$

How about this?

$$(\perp \rightarrow \top) \rightarrow \perp \leq (\alpha \rightarrow \perp) \sqcup \alpha$$

Yes, it turns out, by **case analysis** on  $\alpha$ .

And *only* by case analysis.

# Extensibility

Let's add a new type of function  $\tau_1 \overset{\circ}{\rightarrow} \tau_2$ .

# Extensibility

Let's add a new type of function  $\tau_1 \overset{\circ}{\rightarrow} \tau_2$ .

It's a supertype of  $\tau_1 \rightarrow \tau_2$

*“function that may have side effects”*

# Extensibility

Let's add a new type of function  $\tau_1 \overset{\circ}{\rightarrow} \tau_2$ .

It's a supertype of  $\tau_1 \rightarrow \tau_2$

*“function that may have side effects”*

Now we have a counterexample:

$$\alpha = (\top \overset{\circ}{\rightarrow} \perp) \overset{\circ}{\rightarrow} \perp$$

# Extensibility and vacuous reasoning

$\tau_r$  = some record type

$\tau_f$  = some function type

Is  $\tau_r \sqcap \tau_f \leq \text{bool}$ ?

# Extensibility and vacuous reasoning

$\tau_r =$  some record type

$\tau_f =$  some function type

Is  $\tau_r \sqcap \tau_f \leq \text{bool}$ ?

Yes, vacuously.



# Extensible type systems

Two techniques give us an extensible system:

- Add explicit type variables as indeterminates  
*gets rid of case analysis*

# Extensible type systems

Two techniques give us an extensible system:

- Add explicit type variables as indeterminates  
*gets rid of case analysis*
- Require a distributive lattice  
*gets rid of vacuous reasoning*

# Resulting types

We end up with all the standard types

# Resulting types

We end up with all the standard types  
... with the same subtyping order

# Resulting types

- We end up with all the standard types
- ... with the same subtyping order
- ... but we identify fewer of the weird types

$$\{\text{foo} : \text{bool}\} \sqcap (\top \rightarrow \top) \not\leq \text{bool}$$

# Type Inference with Polar Types

---

# Input and output types

$\tau \sqcup \tau'$ : produces a value which is a  $\tau$  or a  $\tau'$

$\tau \sqcap \tau'$ : requires a value which is a  $\tau$  and a  $\tau'$

$\sqcup$  is for outputs, and  $\sqcap$  is for inputs.

# Input and output types

$\tau \sqcup \tau'$ : produces a value which is a  $\tau$  or a  $\tau'$

$\tau \sqcap \tau'$ : requires a value which is a  $\tau$  and a  $\tau'$

$\sqcup$  is for outputs, and  $\sqcap$  is for inputs.

Divide types into

- *output types*  $\tau^+$
- *input types*  $\tau^-$



# Polar types

$$\tau^+ ::= \text{bool} \mid \tau_1^- \rightarrow \tau_2^+ \mid \{\ell_1 : \tau_1^+, \dots, \ell_n : \tau_n^+\} \mid \\ \alpha \mid \tau_1^+ \sqcup \tau_2^+ \mid \perp \mid \mu\alpha.\tau^+$$

$$\tau^- ::= \text{bool} \mid \tau_1^+ \rightarrow \tau_2^- \mid \{\ell_1 : \tau_1^-, \dots, \ell_n : \tau_n^-\} \mid \\ \alpha \mid \tau_1^- \sqcap \tau_2^- \mid \top \mid \mu\alpha.\tau^-$$

# Cases of unification

In HM inference, unification happens in three situations:

- Unifying two input types
- Unifying two output types
- Using the output of one expression as input to another

# Cases of unification

In HM inference, unification happens in three situations:

- Unifying two input types

*Introduce*  $\sqcap$

- Unifying two output types

*Introduce*  $\sqcup$

- Using the output of one expression as input to another

$\tau^+ \leq \tau^-$  *constraint*

# Eliminating variables, ML-style

Suppose we have an identity function

$$\alpha \rightarrow \alpha$$

# Eliminating variables, ML-style

Suppose we have an identity function, which uses its argument as a  $\tau$

$$\alpha \rightarrow \alpha \mid \alpha = \tau$$

# Eliminating variables, ML-style

Suppose we have an identity function, which uses its argument as a  $\tau$

$$\begin{aligned} & \alpha \rightarrow \alpha \mid \alpha = \tau \\ \equiv^{\forall} & \tau \rightarrow \tau \end{aligned}$$

# Eliminating variables, ML-style

Suppose we have an identity function, which uses its argument as a  $\tau$

$$\begin{aligned} & \alpha \rightarrow \alpha \mid \alpha = \tau \\ \equiv^{\forall} & \tau \rightarrow \tau \end{aligned}$$

The substitution  $[\tau/\alpha]$  solves the constraint  $\alpha = \tau$

# Eliminating variables, MLsub-style

Suppose we have an identity function, which uses its argument as a  $\tau^-$ .

$$\alpha \rightarrow \alpha \mid \alpha \leq \tau^-$$



# Eliminating variables, MLsub-style

Suppose we have an identity function, which uses its argument as a  $\tau^-$ .

$$\begin{aligned} & \alpha \rightarrow \alpha \mid \alpha \leq \tau^- \\ \equiv^{\forall} & (\alpha \sqcap \tau^-) \rightarrow \alpha \end{aligned}$$

# Eliminating variables, MLsub-style

Suppose we have an identity function, which uses its argument as a  $\tau^-$ .

$$\begin{aligned} & \alpha \rightarrow \alpha \mid \alpha \leq \tau^- \\ \equiv^{\forall} & (\alpha \sqcap \tau^-) \rightarrow \alpha \end{aligned}$$

The *bisubstitution*  $[\alpha \sqcap \tau^- / \alpha^-]$  solves  $\alpha \leq \tau^-$

# Decomposing constraints

We only need to decompose constraints of the form  $\tau^+ \leq \tau^-$ .

$$\tau_1 \sqcup \tau_2 \leq \tau_3 \quad \equiv \quad \tau_1 \leq \tau_3, \tau_2 \leq \tau_3$$

$$\tau_1 \leq \tau_2 \sqcap \tau_3 \quad \equiv \quad \tau_1 \leq \tau_2, \tau_1 \leq \tau_3$$

Thanks to the input/output type distinction, the hard cases of  $\tau_1 \sqcap \tau_2 \leq \tau_3$  and  $\tau_1 \leq \tau_2 \sqcup \tau_3$  can never come up.

# Combining solutions

We solve a system of multiple constraints  $C_1, C_2$  by:

- Solving  $C_1$ , giving a bisubstitution  $\xi$
- Applying that to  $C_2$
- Solving  $\xi C_2$ , giving a bisubstitution  $\zeta$

Then  $\xi \circ \zeta$  solves the system  $C_1, C_2$ .

# Summary

MLsub infers types by walking the syntax of the program, but must deal with subtyping constraints rather than just equalities. Thanks to:

- algebraically well-behaved types
- polar types, restricting occurrences of  $\sqcup$  and  $\sqcap$
- the biunification algorithm

we can always handle these constraints, producing a principal type.

# Future work

{-# LANGUAGE ?? #-}

- RankNTypes?
- GADTs?
- TypeFamilies?
- ...

Questions?

<http://www.cl.cam.ac.uk/~sd601/mlsub>  
stephen.dolan@cl.cam.ac.uk

# Mutable references

References are generally considered “invariant”.

Instead, consider `ref` a two-argument constructor

$$(\alpha, \beta) \text{ ref}$$

with operations:

$$\text{make} : (\alpha, \alpha) \text{ ref}$$
$$\text{get} : (\perp, \beta) \text{ ref} \rightarrow \beta$$
$$\text{set} : (\alpha, \top) \text{ ref} \rightarrow \alpha \rightarrow \text{unit}$$